PAPER

Predicate Argument Structure Analysis for Use Case Description Modeling

Hironori TAKEUCHI[†,††a)], *Member*, Taiga NAKAMURA[††], *Nonmember*, *and* Takahira YAMAGUCHI[†], *Member*

SUMMARY In a large software system development project, many documents are prepared and updated frequently. In such a situation, support is needed for looking through these documents easily to identify inconsistencies and to maintain traceability. In this research, we focus on the requirements documents such as use cases and consider how to create models from the use case descriptions in unformatted text. In the model construction, we propose a few semantic constraints based on the features of the use cases and use them for a predicate argument structure analysis to assign semantic labels to actors and actions. With this approach, we show that we can assign semantic labels without enhancing any existing general lexical resources such as case frame dictionaries and design a less language-dependent model construction architecture. By using the constructed model, we consider a system for quality analysis of the use cases and automated test case generation to keep the traceability between document sets. We evaluated the reuse of the existing use cases and generated test case steps automatically with the proposed prototype system from real-world use cases in the development of a system using a packaged application. Based on the evaluation, we show how to construct models with high precision from English and Japanese use case data. Also, we could generate good test cases for about 90% of the real use cases through the manual improvement of the descriptions based on the feedback from the quality analysis system.

key words: *use case, modeling, predicate argument structure, semantic constraint*

1. Introduction

In a large software system development project, many documents are prepared and updated frequently. In such a situation, it is needed to support for quickly looking through these documents easily to identify inconsistencies. For example, by collecting actors in the artifacts describing a specific behavior, we can find which actors should be considered in the specific behavior. However, it sometimes happens that nouns representing actors are not mentioned properly in some descriptions and different verbs the representing same behavior are used. To get information from documents effectively, we have to identify actors and actions and assign semantic information to them in advance.

In some projects, test planning starts in the requirement phases. In such projects, the test cases are created from the requirements documents in the early phases. Through this, we can estimate in advance the resources needed for testing and execute the tests effectively. However, the requirements documents may be updated frequently and we have to update the corresponding test cases to retain their traceability when the requirements are changed to keep the traceability between requirements and tests. Maintaining the traceability between documents manually is difficult and can cause critical problems in the later phases.

In this situation, one should consider extracting information from the texts in the requirements documents and using that information for the software system development. In this research, we focus on the requirements documents such as use cases and consider ways to extract information from the use case descriptions in a free text format and how to use the extracted information. We consider a system to construct models from the use case descriptions by analyzing the predicate argument structures extracted from dependency analysis of use case descriptions. In the modeling, we extract actors and actions and assign semantic labels to them. Semantic labels represent the roles of actors and the types of actions in use case descriptions.

We propose a model construction method that works for the free-text use case descriptions and is less language-dependent. Case frame dictionaries have been used to assign the semantic information in the nouns and verbs in the sentences. Usually, in the case frame dictionaries, "Agent" is assigned for a case with the subject role for the specific verbs, but we have to assign more specific semantics such as "SYSTEM" or "USER" for constructing models. Also we have to assign semantic labels to actions by considering each actor's semantic labels. In the use case description modeling, we found that we cannot apply the existing case frame resources. We therefore introduced language-independent semantic constraints based on the features of the use case and used them for the predicate argument structure analysis to assign semantic labels to actors and actions.

As a result, we found that we can accurately construct the model from the descriptions and can easily apply the proposed model construction approach to other languages. Also, we consider an application using the use case description models. In this research, we consider a requirements analysis system that suggests to a user how to improve the descriptions through the model-based requirements analysis and automatically generates test cases from the use case descriptions.

Organization of this paper: We start by mentioning related work in Sect. 2 and describing the models of the use case descriptions in Sect. 3. In Sect. 4 we show that it is

Manuscript received November 15, 2011.
Manuscript revised March 8, 2012.

[††]The authors are with IBM Research - Tokyo, Tokyo, 135–8511 Japan.
[†]The authors are with Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: hironori@jp.ibm.com

DOI: 10.1587/transinf.E95.D.1959

difficult to apply the conventional approaches for this modeling and propose a method based on semantic constraints. In Sect. 5, we describe our application using the use case description model. In Sect. 6, we evaluate how correctly the model can be constructed from the use case descriptions and how effectively we can use the model in the software development process. After a discussion in Sect. 7, we conclude the paper in Sect. 8.

2. Related Work

There have been some research projects where text analytic technologies were applied to the text of requirement documents. In [1], [2], the text analytics technologies were used to detect ambiguous expressions in the requirements documents. In system or software development, it is necessary to have the same understanding about the important concepts in the target domains. To define the list of terms in an early phase, extracting domain-specific keywords automatically from the requirements documents was considered in [3], [4]. [5]–[7] considered ways to construct domain knowledge from the information extracted from the texts as an ontology describing the relationships among entities, and this knowledge was used to identify inconsistencies in the requirements documents.

In this research, we studied how to construct a model from the scenarios described in the free-text use case descriptions. The related research work, [8] considered how to generate exceptional scenarios from a normal scenario by constructing models from the requirements. In [8], the requirements descriptions are considered to be written in a specific language and not in free-form text. From the free text use case descriptions, constructing a model was studied in [9]. One of the benefits of using models constructed from the texts is that language-independent analysis can be applied. However, in the previous research, the model construction part was strongly language-dependent and cannot easily be used with other languages.

3. Use Case Description Model

Ever since they were introduced by Jacobson in [10], *use cases* have enjoyed popularity among business analysts (BA) and requirement analysts (RA). BAs and RAs like use cases because of their modularity, simplicity and user centric approach. In practice, mostly due to the need of BAs and RAs to interact continuously with their customers, use case descriptions are written in natural language.

Though it is easy to write use case descriptions based on the communications with the customers, it is important that the use case descriptions should be interpreted in similar ways by both developers and users, however simple the descriptions appear to be. To avoid misunderstandings about use case descriptions, some guidelines are described in [11].

In this research, we studied how to construct a model from the use case descriptions to automate some parts of the use case analysis and to support the experts in their work. For the model, we use the Use Case Description (UCD) Model shown in Fig. 1. In a UCD model, each use case description is represented as an Application Model. An Application Model consists of a Domain Model and a Use Case Model. The Domain Model describes the actors appearing in the use case. For each actor, a semantic label "SYSTEM" or "USER" will be assigned. The Use Case Model describes the sequences of sentences that contain actions between users and the system. Each action consists of the initiating actor and an action verb with an action type representing the semantics of the action, such as "INPUT", "CREATE", "UPDATE" or "OUTPUT", etc. In this research, we focus on the main flows in use cases. In many projects, use cases are described in the project-specific structures and the flows for exceptions are described in the separated sections in use case documents. In such cases, we can apply our method to the descriptions in the section of exceptional flows.

Figure 2 shows an example of a use case description using the UCD model. Note that in this example, "PRO_USR", "PRO_ADM", and "PRO_SYS" are the actor names. In a project, many project-specific terms representing actors are sometimes defined and it is important to properly understand the semantic types of such actor names. In our research, we will assign the semantic labels for such project-specific actor names.

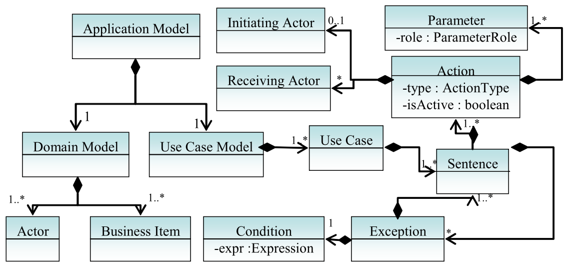
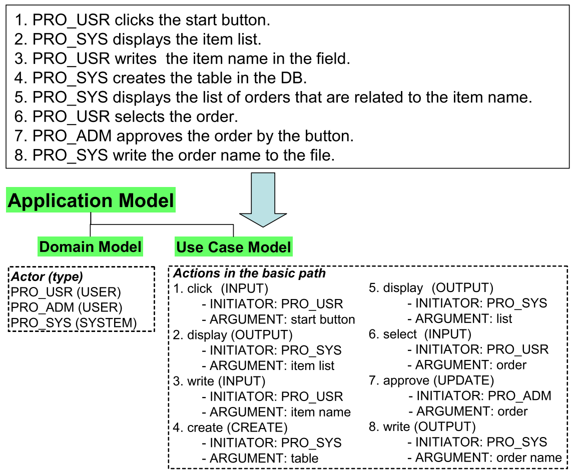


Fig. 1 Use case description model.


1. PRO_USR clicks the start button.
2. PRO_SYS displays the item list.
3. PRO_USR writes the item name in the field.
4. PRO_SYS creates the table in the DB.
5. PRO_SYS displays the list of orders that are related to the item name.
6. PRO_USR selects the order.
7. PRO_ADM approves the order by the button.
8. PRO_SYS write the order name to the file.

Application Model

Domain Model

Actor (type)
PRO_USR (USER)
PRO_ADM (USER)
PRO_SYS (SYSTEM)

Use Case Model

Actions in the basic path
1. click (INPUT)
   - INITIATOR: PRO_USR
   - ARGUMENT: start button
2. display (OUTPUT)
   - INITIATOR: PRO_SYS
   - ARGUMENT: item list
3. write (INPUT)
   - INITIATOR: PRO_USR
   - ARGUMENT: item name
4. create (CREATE)
   - INITIATOR: PRO_SYS
   - ARGUMENT: table
5. display (OUTPUT)
   - INITIATOR: PRO_SYS
   - ARGUMENT: list
6. select (INPUT)
   - INITIATOR: PRO_USR
   - ARGUMENT: order
7. approve (UPDATE)
   - INITIATOR: PRO_ADM
   - ARGUMENT: order
8. write (OUTPUT)
   - INITIATOR: PRO_SYS
   - ARGUMENT: order name

Fig. 2 Example of a use case description using the UCD model.

4. Use Case Description Model Construction

4.1 Model Construction Overview

In this research, as a precondition, we assume the input is natural language texts describing the use case. Since our approach depends on the outputs of underlying linguistic analyzers, such as a morphological analyzer and a dependency parser, erroneous inputs to these analyzers often result in an incorrect result. For the same reason, while our approach does not explicitly restrict how the use cases should be written, the level of complexity and rigidity of the description affects the accuracy of these analyzers. Here are the steps for constructing a model:

1. Lexical and dependency information extraction
2. Anaphora resolution and clause identification
3. Extraction of predicate argument structures from the main clauses.
4. Assign semantic information for the verb and the subject in the extracted predicate argument structure.

Steps 1, 2, and 3 are language-dependent, but there are widely used public analyzers available for most prominent languages. By using the UIMA framework [12] with which linguistic analysis engines can be used as plug-ins, we can easily switch the analysis engines when the language of the data is changed. We can therefore get the predicate argument structures in several major languages by using off-the-shelf lexical processors. By constructing an anaphora resolver and a clause detector for each language, we can extract predicate argument structures from the use case descriptions. The predicate argument structure is the lexical information about the arguments of a verbal predicate with their syntactic properties. Figure 3 shows a predicate argument structure extracted through Step 1, 2 and 3.

In Step 4, we have to assign semantic labels to the verbs and the noun phrases, each of whose grammatical role is a subject from each extracted predicate argument structure. We describe the details of this step in the next subsections.

4.2 Model Construction from Predicate Argument Structures Using Case Frame Resources

One approach for assigning domain-specific semantic labels is to use a dictionary. This approach is simple and practical. For example, by defining "user" and "administrator" in the dictionary we can extract the compound nouns whose head word is one of the registered words. As a result, we can assign the semantic label "USER" to compound nouns such as "tool user" and "system administrator". In real use cases, however, there are some domain specific variations representing systems and users as shown in Fig. 2 and we cannot prepare dictionaries that can be widely used in advance. In such situations, we have to add domain-specific terms representing systems and users and enhance the dictionary for each data set. Also, we have to assign semantic labels to verbs, but the semantic label for the verb sometimes depends on the actor. For example, as described in Fig. 2, the label for "write" is "INPUT" or "OUTPUT" when the actor is "USER" or "SYSTEM", respectively. Those semantic labels are determined from the point of the user's view. In these situations, we have to enhance the dictionary but the enhancement will be complicated because we have to consider the contexts, which make the enhancement expensive.

To assign semantic labels to cases in the predicate argument structures, case frame dictionaries are generally used. For each verb, the cases that the verb should cover and their corresponding deep cases are defined in the case frame dictionary[†]. For the deep cases, 8 cases such as "Agent" and "Location" are defined and these are language-dependent. In the case frame dictionary mentioned above, in 9144 out of 10364 verbs, "Agent" is assigned to the case whose grammatical role is a subject. "Agent" corresponds to "Actor" in the use case descriptions but we want to assign a more specific semantic label such as "SYSTEM" or "USER" when constructing a model. In some digital lexical resources, semantic features such as "human" or "organization" are assigned to the deep cases for each verb. For example, in the basic verb dictionary in IPA lexicon, 19 semantic features are defined [13]. In the use cases, however, the system's behaviors tend to be described in an anthropomorphic way. For examples, in use cases, "write", "read", "record", "delete" and "use" are used in the descriptions of the behaviors of both SYSTEM and USER but "human" or "human or organization" is assigned to those verbs as the semantic feature of the agent case in the IPA Lexicon. In the Process Handbook [14], the more detailed knowledge on typical business activities is defined by focusing on *what* (Object Case) and *how* (Implement Case). However in the statements 3 and 8 in Fig. 2, same type of concepts are described with the object case and we cannot use this knowledge of the business activities to determine the action and the actor type. Therefore, we need to enhance these resources for the use case domain if we want to assign semantic labels for actors and actions. Generally, experts must enhance the linguistic resources at great expense to maintain these resources for each language. In the next subsection, we propose an approach that does not depend on case frame resources.

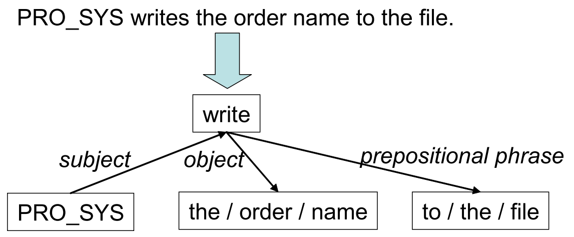


Fig. 3 Example of an extracted predicate argument structure.

[†]For example, a Japanese predicate argument structure thesaurus is publicly available at
http://cl.it.okayama-u.ac.jp/rsc/data/index.html

4.3 Model Construction from Predicate Argument Structures Using Use Case Oriented Semantic Constraints

As mentioned above, it is difficult to construct models from use case descriptions by using case frame resources if the rich information on the deep cases is available. In this work, we propose a model construction approach using semantic constraints based on the features of the use cases. In the semantic constraints, conditions to determine the semantic roles of actions or actors are defined. In the use cases, we describe the interactions between the user and the system from the user's viewpoint. We can therefore consider that use case descriptions satisfy these features:

- The semantic label of the actor is "USER" or "SYSTEM".
- Verbs used in the use case descriptions can be covered by the verb dictionary prepared in advance.
- Specific actions such as "INPUT" or "OUTPUT" are described in most use cases.

First, by using the dictionary, we assign initial semantic labels to the verbs. As the initial verb semantic labels, we start with INPUT, OUTPUT, READ, WRITE, GIVE, GET, CREATE, QUERY, UPDATE, DELETE, USE, START, STOP, and BROWSE. Verbs that do not have corresponding entries in the dictionary are annotated as UNCLASSIFIED. Verb dictionaries are language dependent but we can easily construct them from the corpus. For this research we quickly constructed the dictionaries by using existing documents. We collected frequent 200 verbs for English and Japanese from requirements documents in various domains. For each verb semantic label, we assigned verbs from the collected verb list. Figure 4 shows the sample of the dictionary for English. Figure 5 shows the description when the verb dictionary is used.

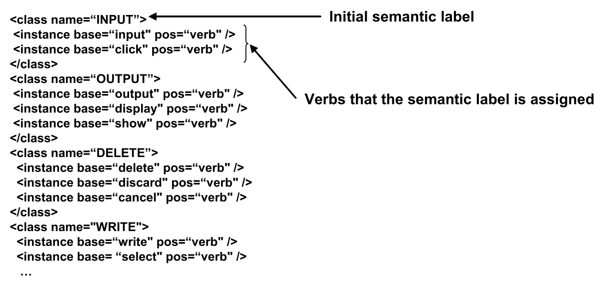
Now, we try to determine the semantic labels for subjects and verbs. For the predicate argument structure where the verb has an initial semantic label, we apply two types of semantic constraints, "Verb-To-Subject" constraints and "Subject-To-Verb" constraints. The Verb-To-Subject semantic constraints determine the semantic label of the subject (actor) with its confidence score s. The Subject-To-Verb semantic constraints determine the semantic label of the verb from the specified semantic label of the actor and the initial verb semantic label.

From the features of the use case descriptions, we can expect that most use case descriptions contain sentences representing an "INPUT" or "OUTPUT" action. In such sentences, the semantic label of the subjects are determined to "USER" or "SYSTEM" respectively. On the other hand, as mentioned in Sect. 4.2 some semantic labels for verbs depend on the semantic labels of their subjects. From those two types of constraints, therefore we apply the Verb-To-Subject semantic constraints first. We defined 14 constraints on the semantic label of the subject for the specific verb types. Here are some examples:

- Initial verb semantic label = INPUT $\rightarrow$ Subject semantic label = USER, $s = 1.0$
- Initial verb semantic label = OUTPUT $\rightarrow$ Subject semantic label = SYSTEM, $s = 1.0$
- Initial verb semantic label = START $\rightarrow$ Subject semantic label = SYSTEM, $s = 0.5$
- Initial verb semantic label = CREATE, READ, UPDATE, DELETE $\rightarrow$ Subject semantic label = USER, $s = 0.1$

In each constraint, the left side of the arrow represents the condition and the right side of the arrow represents the assigned label to the subject and the confidence s. The confidence takes the value from 0.0 to 1.0. For example, when the initial verb semantic label is "INPUT", the label for the actor is determined as "USER" with its confidence score s=1.0. The full set of the Verb-To-Subject semantic constraints is shown in Appendix (Table A· 1). If a predicate argument structure satisfies a constraint with $s = 1.0$, the subject semantic label defined in the constraint is assigned and the other constraints are not applied. In other cases, for each noun that appeared in the subject case, we get the confidence scores by applying constraints to predicate argument structures. After comparing the confidence scores on both SYSTEM and USER, the subject semantic labels are determined. Figure 6 shows the results by applying the Verb-To-Subject constraints and how the semantic labels for subjects are determined.

Next we apply the Subject-To-Verb semantic constraints. We defined 9 constraints on the semantic labels of the verbs for the subject types. With these constraints, some initial verb semantic labels are changed to their final verb semantic labels. As the final verb semantic labels, we defined 9 labels, INPUT, OUTPUT, QUERY, CREATE, UP-

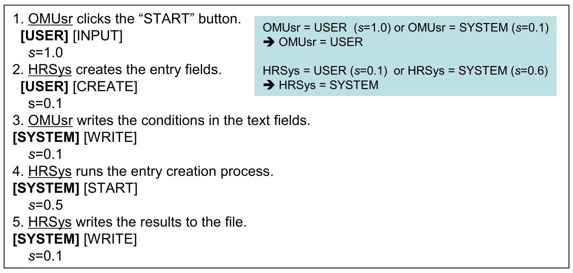


```
<class name="INPUT">
 <instance base="input" pos="verb" />
 <instance base="click" pos="verb" />
</class>
<class name="OUTPUT">
 <instance base="output" pos="verb" />
 <instance base="display" pos="verb" />
 <instance base="show" pos="verb" />
</class>
<class name="DELETE">
  <instance base="delete" pos="verb" />
  <instance base="discard" pos="verb" />
  <instance base="cancel" pos="verb" />
</class>
<class name="WRITE">
  <instance base="write" pos="verb" />
  <instance base= "select" pos="verb" />
  ...
```

Fig. 4 Example of entries in the verb dictionary.

1.OMUsr <u>clicks</u> the "START" button.
[INPUT]
2. HRSys <u>creates</u> the entry fields.
[CREATE]
3. OMUsr <u>writes</u> the id and the conditions in the text fields.
[WRITE]
4. HRSys <u>runs</u> the entry creation process.
[START]
5. HRSys <u>writes</u> the results to the file.
[WRITE]

Fig. 5 Example of the description that verb dictionary look up is applied.

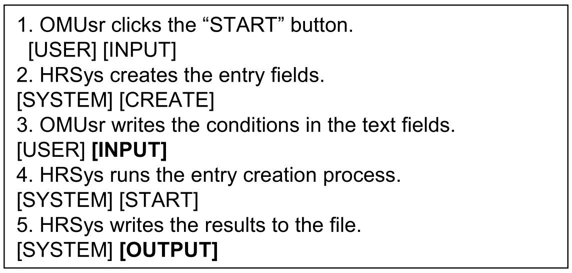


1. OMUsr clicks the "START" button.
[USER] [INPUT]
s=1.0
2. HRSys creates the entry fields.
[USER] [CREATE]
s=0.1
3. OMUsr writes the conditions in the text fields.
[SYSTEM] [WRITE]
s=0.1
4. HRSys runs the entry creation process.
[SYSTEM] [START]
s=0.5
5. HRSys writes the results to the file.
[SYSTEM] [WRITE]
s=0.1

OMUsr = USER (s=1.0) or OMUsr = SYSTEM (s=0.1)
➔ OMUsr = USER

HRSys = USER (s=0.1) or HRSys = SYSTEM (s=0.6)
➔ HRSys = SYSTEM

Fig. 6 Example of the description that the verb-to-subject constraints are applied.



1. OMUsr clicks the "START" button.
[USER] [INPUT]
2. HRSys creates the entry fields.
[SYSTEM] [CREATE]
3. OMUsr writes the conditions in the text fields.
[USER] **[INPUT]**
4. HRSys runs the entry creation process.
[SYSTEM] [START]
5. HRSys writes the results to the file.
[SYSTEM] **[OUTPUT]**

Fig. 7 Example of the description that the subject-to-verb constraints are applied.

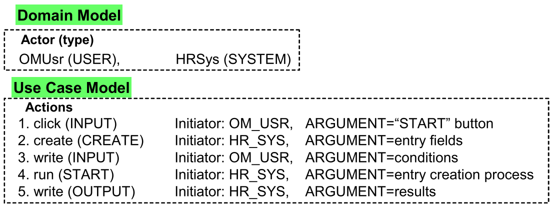
DATE, DELETE, START, STOP, BROWSE. Here are some examples:

- Subject semantic label = SYSTEM (or UNCLASSIFIED) & Initial verb semantic label = WRITE → Final verb semantic label = OUTPUT
- Subject semantic label = USER & Initial verb semantic label = WRITE → Final verb semantic label = INPUT
- Subject semantic label = SYSTEM (or UNCLASSIFIED) & Initial verb semantic label = USE → Final verb semantic label = QUERY

In each constraint, the left side of the arrow represents the condition on the subject semantic label and the initial verb semantic label and the right side of the arrow represents the finally assigned label for the verb. For example, when the subject semantic label is "SYSTEM" and the initial verb semantic label is "WRITE", the label for the verb is determined as "OUTPUT". If no constraints are matched, the initial verb semantic labels are mapped to the final verb semantic labels unchanged. Figure 7 shows the results by applying the Subject-To-Verb constraints.

After applying those constraints, any semantic labels including "UNCLASSIFIED" are assigned to the subject and the verb in each predicate argument structure representing an action. As a result, we can construct a model for the input use case description (Fig. 8).

Those constraints are language independent. We can therefore apply those constraints in the predicate argument structure analysis for any language data. In this proposed method, we have to implement some small language-dependent modules and prepare the verb dictionaries. But compared to enhancing the case grammars for each language, these efforts are relatively inexpensive. As a result, we can easily apply this method for the use case modeling to other languages.



Domain Model
Actor (type)
OMUsr (USER), HRSys (SYSTEM)

Use Case Model
Actions
1. click (INPUT) Initiator: OM_USR, ARGUMENT="START" button
2. create (CREATE) Initiator: HR_SYS, ARGUMENT=entry fields
3. write (INPUT) Initiator: OM_USR, ARGUMENT=conditions
4. run (START) Initiator: HR_SYS, ARGUMENT=entry creation process
5. write (OUTPUT) Initiator: HR_SYS, ARGUMENT=results

Fig. 8 Example of the constructed model.

5. Application

In this section, we describe our application using the use case description models.

5.1 Application Background

Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM) systems are packaged applications supporting standard business functions and processes. To deploy such systems, they need to be configured and sometimes customized to meet the unique needs of the businesses. In such cases, many use cases are prepared for a project and these use cases can be reused within the same project and across projects. To reuse the use cases effectively, a system for assisting business consulting is proposed in [15]. In this system, existing use cases are categorized into the known business process hierarchy[†] and stored on a server. Consultants access the business hierarchy from their client machines and can list all of the requirements in a given project.

In such cases, before storing the use cases into the repository, we have to review them and improve their quality so that every user can have the same understanding of each use case. Therefore, detecting the problems in the description through the models and providing the recommendations to users are useful. Also, if we can automatically transform the user case descriptions into other artifacts such as test cases, then we can easily update those artifacts when requirements are changed.

In this section we consider a system for the model-based use case quality analysis and test case template generation.

5.2 Model-based Use Case Quality Analysis

Here, we describe ours model-based use case quality analysis system.

The constructed models can help identify these kinds

[†]For example, the American Productivity and Quality Center (APQC) provides Process Classification Framework (http://www.apqc.org/process-classification-framework).

of errors:

- Actor is missing.
- Actor cannot be classified.
- Action cannot be classified.

These errors come from the guideline that says we should describe the interactions between the user and the system precisely [1], [11].

Also, these cases can be identified as warnings:

- There are multiple actions described in a use case statement.
- The voice of the statement is passive.

Those items come from the guideline that we should describe the use cases as simply as possible [11]. In this quality analysis, parts of the unambiguity and correctness considered in [16] can be covered.

Figure 9 shows the results of the use case description analysis system. In this system, the user writes the use case description in the editor. When the description is saved, the model is automatically constructed from the text. As a result, on the editor, the actors and action verbs are annotated based on the model. Results of the model-based quality analysis are shown as recommendations in the upper pane in the editor. In Fig. 9, an error and two warnings are identified by the system. When a user changes the descriptions and saves the updates, the model construction and the model analysis are updated.

5.3 Test Case Template Generation

As the other application, we show how the system generates the test case from the use case. In this research, we consider that the test cases are scripts representing what one should do for testing and what should be checked. In real projects, the test cases for user testing are prepared from the use cases. For user testing, the steps that the test engineers should perform (execution steps) and the steps describing the expected results (verification steps) have to be written for the test case. Here is how these steps are generated from each statement in the use case in the application:

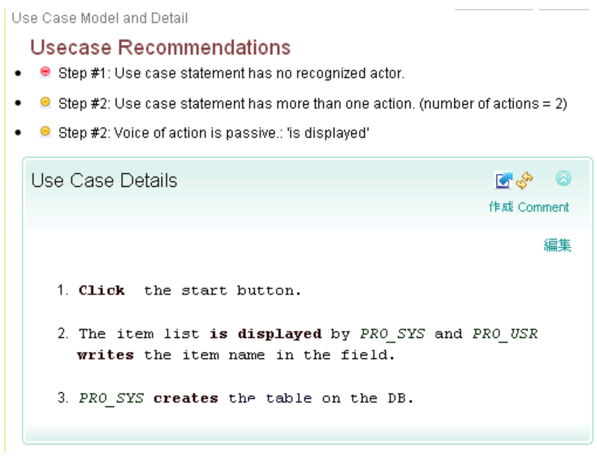
- If the initiating actor represents "USER", then an execution step is generated.
- If the initiating actor represents "SYSTEM", then a verification step is generated.

For an execution step, the actor description is removed from the use case statement and the step description is generated in the imperative form. For a verification step, the step description is generated in a form such as "Verify that [Action description]".

Figure 10 shows the result of the test case template generation system. Input is the use case description in Fig. 2 and the test case is successfully generated. Each statement in the use case description is properly categorized to the execution or the verification step because the actor semantic labels are properly assigned.

When actor is missing or actor cannot be classified, the test case template generation system will not work properly. For these cases, the generation system writes the original sentence in the execution step column. In the model construction, we can construct the action model for the system's operation to a database. The test engineers cannot verify such a system's action. In this case, the generation system also writes the original sentence in the execution step column. Figure 11 shows an example when the test case was not generated correctly.

In this application, we can expect that the modifiability and the traceability [16] between use cases and test cases are improved. In many projects, requirements are updated based on the feedback from the implementation. If we manually prepare test cases from use cases before their implementation, it is difficult to keep the consistencies between use cases and manually created test cases. Therefore the test phases are usually started when the implementation is completed and all requirements are fixed. Using this test case generation application, we can start the preparation of the



Use Case Model and Detail

Usecase Recommendations

- Step #1: Use case statement has no recognized actor.
- Step #2: Use case statement has more than one action. (number of actions = 2)
- Step #2: Voice of action is passive.: 'is displayed'

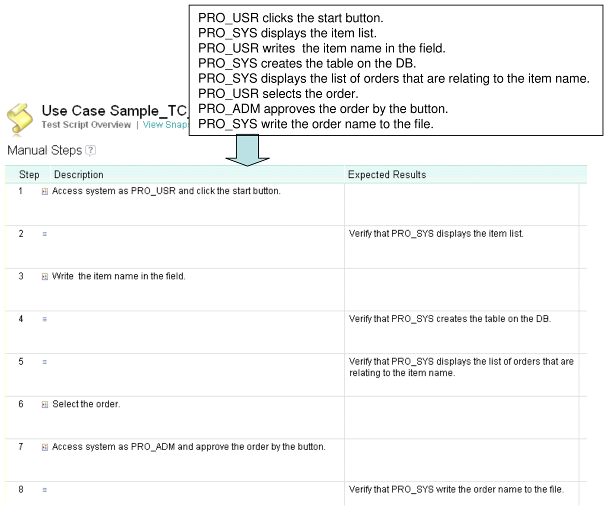
Use Case Details

作成 Comment

編集

1. **Click** the start button.
2. The item list **is displayed** by *PRO_SYS* and *PRO_USR* **writes** the item name in the field.
3. *PRO_SYS* **creates** the table on the DB.

Fig. 9 Example of use case quality analysis.



PRO_USR clicks the start button.
PRO_SYS displays the item list.
PRO_USR writes the item name in the field.
PRO_SYS creates the table on the DB.
PRO_SYS displays the list of orders that are relating to the item name.
PRO_USR selects the order.
PRO_ADM approves the order by the button.
PRO_SYS write the order name to the file.

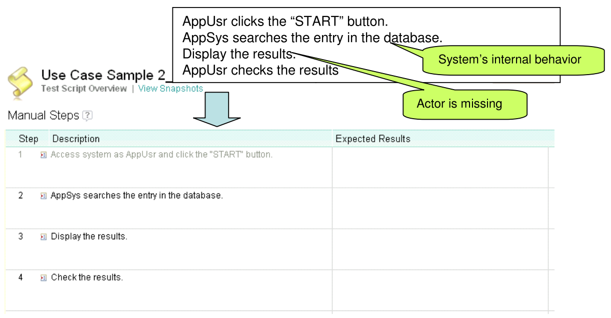
Use Case Sample_TC

Test Script Overview | View Snap

Manual Steps

Step	Description	Expected Results
1	Access system as PRO_USR and click the start button.	
2		Verify that PRO_SYS displays the item list.
3	Write the item name in the field.	
4		Verify that PRO_SYS creates the table on the DB.
5		Verify that PRO_SYS displays the list of orders that are relating to the item name.
6	Select the order.	
7	Access system as PRO_ADM and approve the order by the button.	
8		Verify that PRO_SYS write the order name to the file.

Fig. 10 Example of the generated test case (Success).


System's internal behavior
Actor is missing
Use Case Sample 2
Test Script Overview | View Snapshots
Manual Steps

AppUsr clicks the "START" button.
AppSys searches the entry in the database.
Display the results.
AppUsr checks the results

Step	Description	Expected Results
1	Access system as AppUsr and click the "START" button.	
2	AppSys searches the entry in the database.	
3	Display the results.	
4	Check the results.	

Fig. 11 Example of the generated test case (Failed).

test cases before the system implementation is completed and can easily keep the consistencies when the use cases are changed. As a result, we can reduce the duration needed for testing.

6. Evaluation

First we evaluate how well the proposed method can extract action models from use case descriptions. For this we consider two types of evaluation. One of them is how well the action verb can be identified. We evaluate this by using the standard metrics of precision (P) and recall (R). The other is how well the extracted action models capture the correct information. We evaluate this by T defined in (1)

$$T = 1 - \frac{|A_u|}{|A_c|}, \tag{1}$$

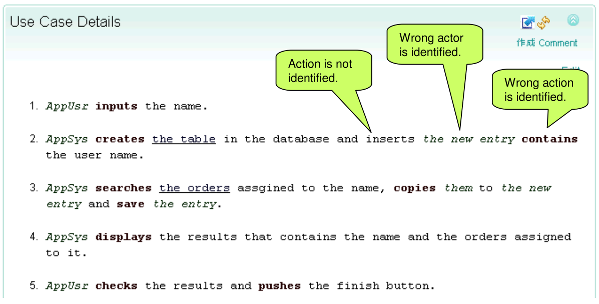
where A_c represents the set of actions and A_u represents the set of extracted actions that miss the information for some model elements. T represents the ratio of information not lost in the model extraction. In the evaluation experiments, we used 80 use cases written in English and 12 use cases written in Japanese. Table 1 shows the evaluation results for the model extraction. The results on P and R show how accurately the predicate argument structures in the main clauses were extracted and how many correct structures were extracted, respectively.

We found that some predicate argument structures were incorrectly extracted in some Japanese and English use cases. Figure 12 shows the description that our model construction method does not work because all predicate argument structures are not extracted properly. In this example, the linguistic analyzer cannot handle the second sentence because "that" representing the relative clause is omitted and this generates an incorrect predicate argument structure.

The results on T show how accurately the proposed method was able to assign the semantic information to actors and action verbs. Compared to the English use case data, the semantic information was not assigned as accurately for the Japanese use case data. For examples, action verbs missed information on the action type and in such cases UNCLASSIFIED was assigned. This shows that some of the verbs appearing in the Japanese data were not covered by the pre defined verb dictionary.

Table 1 Evaluation results on the modeling.

	P	R	T
English Data	0.967	0.972	0.930
Japanese Data	0.815	0.946	0.765


Use Case Details
作成 Comment
Action is not identified.
Wrong actor is identified.
Wrong action is identified.

1. *AppUsr* **inputs** the name.
2. *AppSys* **creates** the table in the database and inserts *the new entry* **contains** the user name.
3. *AppSys* **searches** the orders assgined to the name, **copies** *them* to *the new entry* and **save** *the entry*.
4. *AppSys* **displays** the results that contains the name and the orders assigned to it.
5. *AppUsr* **checks** the results and **pushes** the finish button.

Fig. 12 Example of the use case that the model construction does not work well.

Table 2 Result of the test case generation.

	Number of Use Cases	Precision
Data set A	28	28/28 (100%)
Data set B	44	38/44 (86.4%)
All Data	72	66/72 (91.2%)

From the evaluation off the model construction, it is found that we can correctly create the model from use case descriptions with high probability. Next, we consider how effectively we can utilize the constructed model in the software development process. We evaluated the application using the use case description models described in Sect. 5. For the evaluation of the proposed application, we collected 72 real use cases prepared for a packaged application system for the human resources processes. These use cases can be reused in other projects to construct similar systems. Here, we evaluate how use cases can be reused through the proposed application. The average number of statements (sentences) in each use case description is 16.4. For the 72 use case descriptions, we applied the requirement analysis described in Sect. 5.2. As a result, errors or warnings are not detected in 28 use cases. For these 28 use cases (Data set A), we directly used the test case generation process. For the remaining 44 use cases (Data set B), we modified the descriptions in the editor (Fig. 9) based on the feedback from the application and applied the test case generation process. We evaluated whether the execution steps and the verification steps were correctly generated for the test case templates for these two datasets. Table 2 shows the evaluation results.

From these results, we found that in the data set A the test cases were generated successfully, but in the data set B there were some problems that were not detected by the requirement analysis system and test cases were not generated properly from such use cases.

7. Discussion

Through the evaluation we found that the precision of the extraction of the predicate argument structures from our Japanese data was lower than that from the English data. One of the reason was that the relative clauses are not identified correctly in some cases because of excess and misleading text in the use case descriptions such as the colloquial expressions ("〜している" (shite iru)). Currently, in many projects, there are many engineers whose native language is not Japanese. For these engineers it is sometimes difficult to understand Japanese descriptions containing flawed expressions [17]. Such expressions can be removed in preprocessing. This should improve the precision.

For our modeling method, we only need a verb dictionary for each language and do not have to enhance semantic constraints for assigning semantic labels. In this research, we focused on the analysis of main flows in use cases. Usually, the flows for exceptions are described in other sections in the use case documents and we can apply the same approach to the descriptions in such sections. However, if both the main flow and the flow for the exception are written in one description, we have to analyze conditional clauses such as if-statement and when-statement for the model construction. This is language-dependent and we have to prepare a specific analyzer for each language [18].

When applying our method to a domain project, we need to add domain-specific verbs that are not registered in the current dictionary to get semantic information for actions more precisely. This domain adaptation can be realized by extracting frequent verbs from randomly sampled documents in the target domain data. For the model construction from use case descriptions written in English or Japanese, we prepared dictionaries of the same size (200) from the corpus to assign the initial semantic labels to the action verbs. Through the evaluation of the correctness of the extracted model, we found that the coverage of the verb dictionary was not sufficient in Japanese compared to English. This indicates that the number of verbs for adequate coverage of the use case descriptions is language-dependent for some languages. To construct an optimized basic verb dictionary that can be applied to any kind of data, we should investigate more real use case data from various domains. This is one of our future goals.

As target applications for our use case description models, we proposed a requirements analysis system and a test case template generation system. By working with the real use case data, we found that we could generate test cases correctly from about 90% of the real use cases if we tuned the descriptions based on the feedback from the requirements analysis system. Though there were still some use cases with problems that were not detected by the system, our results show that all of the use cases with errors could be covered by the system. Therefore we believe that we can identify almost all of the use cases that contain errors by using the proposed analysis system and we can refine the use cases properly and make them reusable by reviewing all of the descriptions when errors or warnings are detected.

We focused on the requirements analysis and the test case template generation system based on our model. Beyond these applications, we hope to use our models for analyzing use case dependencies. In application development projects, it is necessary to check the dependencies and consistencies between use cases. In such a case, for example, when there is an action in which the system deletes the business item in a use case, we may need to collect all of the use cases in which the business item was created. In some projects, hundreds of use cases are prepared and there are some variations describing the delete and create actions. Therefore it is difficult to collect all the use cases describing a specific action manually. By using our model construction approach, we can easily recognize the use cases mentioning a specific action. Considering other applications and evaluating their effectiveness is also future work.

8. Conclusion

In this research, we focused on the requirement documents such as use cases and consider how to create models from the use case descriptions in a free text format. In the model construction, we showed that we cannot apply the existing case frame dictionaries for the predicate argument structure analysis to assign semantic labels to initiating actors and action verbs. We therefore proposed a method to introduce a few semantic constraints based on the features of the use cases and apply them to the predicate argument structure analysis. We showed that with the proposed method, we can assign semantic labels without using case frame dictionaries and can design a less language-dependent model construction architecture. From our evaluations, we found that we can construct model with high precision from English and Japanese use case data. By using the constructed model, we also showed how the system works for the quality analysis of use cases and how it automates test case generation. Through the real use cases in the system development using the packaged application, we evaluate how we can reuse the existing use cases and generate test case steps automatically with the proposed system. Also, we found that we can generate test cases successfully from about 90% of the real use cases by manually improving the descriptions based on feedback from the quality analysis system.

To create the model more precisely, it is found that we need to investigate the real use case data from various domains and find the approach to prepare the verb dictionary that can be used widely. From the application evaluation, we can reuse the use case effectively. Our results indicate that we can use the model for other purposes such as analyzing use case dependencies. We need to consider other applications and evaluate the effectiveness in the real use in the future.

References

[1] D.M. Berry and E. Kamsties, "Ambiguity in requirements specification," Perspectives on Software Requirements, pp.7–44, 2004.

[2] H. Yang, A. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Extending nucuous ambiguity analysis for anaphora in natural language requirements," Proc. 18th IEEE International Requirements Engineer-

ing Conference, pp.25–34, 2010.

[3] P. Sawyer, P. Rayson, and K. Cosh, "Shallow knowledge as an aid to deep understanding in early phase requirements engineering," IEEE Trans. Softw. Eng., vol.31, no.11, pp.969–981, 2005.

[4] R. Gacitua, P. Sawyer, and V. Gervasi, "On the effectiveness of abstraction identification in requirements engineering," Proc. 18th IEEE International Requirements Engineering Conference, pp.5–14, 2010.

[5] H. Kaiya and M. Saeki, "Ontology based requirement analysis: Lightweight sementic processing approach," Proc. Fifth IEEE International Conference on Quality Software (QSIC), pp.223–230, 2005.

[6] L. Kof, Text Analysis for Requirement Engineering, VDM Verlag, 2007.

[7] R. Hasegawa, M. Kitamura, H. Kaiya, and M. Saeki, "Extracting conceptual graphs from japanese documents for software requirements modeling," Proc. 6-th Asia-Pacific Conference on Conceptual Modeling (APCCM), pp.87–96, 2009.

[8] A. Ohnishi, "A generation method of exceptional scenarios from a normal scenario," ICICE Trans. Inf. Software, no.4, pp.881–887, 2008.

[9] A. Sinha, A. Paradkar, P. Kumanan, and B. Boguraev, "A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases," Proc. IEEE/ACM DSN, pp.327–336, 2009.

[10] I. Jacobson, "Object-oriented software engineering - A use case driven approach," TOOLS, vol.10, p.333, 1993.

[11] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, Boston, MA, USA, 2000.

[12] D. Ferrucci and A. Lally, "UIMA: An architectural approach to unstructured information processing in the corporate research environment," Natural Language Engineering, vol.10, no.3-4, pp.327–348, 2004.

[13] Structure and Format for Digital Dictionaries for Japanese Language Analysis, IPSJ-TS 0003, 2004 (in Japanese).

[14] T.W. Malone, K. Crowston, and G.A. Herman, Organizing Business Knowledge: The MIT Process Handbook, MIT Press, Cambridge, MA, USA, 2003.

[15] P. Mazzoleni, S. Goh, R. Goodwin, M. Bhandar, S.K. Chen, J. Lee, V.S. Sinha, S. Mani, D. Mukherjee, B. Srivastava, P. Dhoolia, E. Fein, and N. Razinkov, "Consultant assistant: A tool for collaborative requirements gathering and business process documentation," Proc. ACM SIGPLAN, pp.807–808, 2009.

[16] IEEE, "Recommended practice for software requirements specifications," ANSI/IEEE Standard 830-1998, 1998.

[17] H. Takeuchi, S. Ogino, T. Nakada, Y. Sakamoto, and N. Fukuoka, "Quality analysis of development-related documents using text analytics technologies," IPSJ Embedded Software Symposium, pp.93–100, 2009 (in Japanese).

[18] H. Takeuchi, T. Nakamura, and T. Yamaguchi, "Use case analysis using text analytics," IEICE Technical Report, KBSE2010-32, 2010 (in Japanese).

Appendix: Semantic Constraints

Here, we show the full set of semantic constraints. Table A· 1 shows the Verb-To-Subject semantic constraints. In this table, for each constraint, the "Initial verb semantic label" column represents the condition and "Subject sematic label" represents the induced semantic label, with the confidence value *s* in the third column.

Table A· 2 shows the Subject-To-Verb semantic constraints. In this table, for each constraint, the "Subject semantic" column and the "Initial verb semantic label" column represent the condition and "Final verb sematic label" represents the induced verb semantic label.

Table A· 1 Verb-to-subject semantic constraints.

Initial verb semantic label	Subject semantic label	*s* (confidence)
INPUT	USER	1.0
OUTPUT	SYSTEM	1.0
START	SYSTEM	0.5
STOP	SYSTEM	0.5
USE	USER	0.5
BROWSE	USER	0.5
CREATE	USER	0.1
READ	USER	0.1
UPDATE	USER	0.1
DELETE	USER	0.1
READ	SYSTEM	0.1
WRITE	SYSTEM	0.1
GIVE	SYSTEM	0.1
GET	SYSTEM	0.1

Table A· 2 Subject-to-verb semantic constraints.

Subject semantic label	Initial verb semantic label	Finel verb semantic label
USER	WRITE	INPUT
USER	GIVE	INPUT
USER	READ	OUTPUT
USER	GET	OUTPUT
SYSTEM(or UNCLASSIFIED)	WRITE	OUTPUT
SYSTEM(or UNCLASSIFIED)	GIVE	OUTPUT
SYSTEM(or UNCLASSIFIED)	READ	INPUT
SYSTEM(or UNCLASSIFIED)	GET	INPUT
SYSTEM(or UNCLASSIFIED)	USE	QUERY

Hironori Takeuchi is a researcher at IBM Research-Tokyo in Japan. He received the B.E. and M.E. degrees from University of Tokyo in 1998 and 2000, respectively. His research interests include text mining, information retrieval, and text analytics in software engineering. He is a member of Information Processing Society (IPS) Japan and Japanese Society for Artificial Intelligence (JSAI).

Taiga Nakamura is a researcher at IBM Research-Tokyo in Japan. His research and industrial activities include software defect patterns, metrics, test automation, software quality engineering and empirical methods in software engineering. He received the B.E. and M.E. degrees in aerospace engineering from University of Tokyo, and received the Ph.D. degree in computer science from University of Maryland, College Park. He is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers, and the Information Processing Society of Japan.

Takahira Yamaguchi is a professor at the Faculty of Science and Technology at Keio University. He received his B.E., M.E., and Ph.D. degrees in telecommunication engineering from Osaka University in 1979, 1981, and 1984, respectively. His research interests include Ontology Engineering, KBSE, Advanced Knowledge Systems, and Machine Learning. He is a member of IPSJ, JSAI, JSFTS, JCSS, ISSJ, AAAI, IEEE-CS, and ACM.